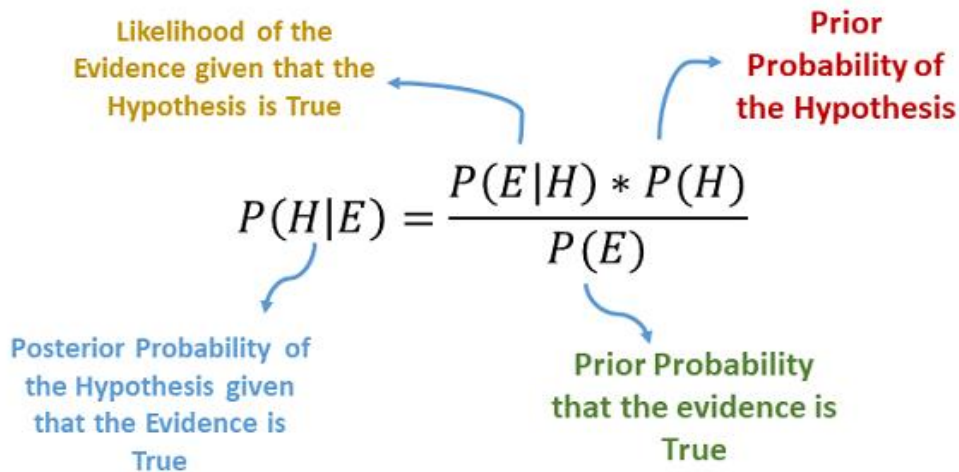


## Naive Bayes Algorithm

Naive Bayes is a classification technique that is based on Bayes' Theorem with an assumption that all the features that predicts the target value are independent of each other. It calculates the probability of each class and then pick the one with the highest probability. It has been successfully used for many purposes, but it works particularly well with natural language processing (NLP) problems.

Bayes' Theorem describes the probability of an event, based on a prior knowledge of conditions that might be related to that event.



The diagram shows the Bayes' Theorem formula with four annotations and arrows:

- Likelihood of the Evidence given that the Hypothesis is True** (yellow text) points to  $P(E|H)$ .
- Prior Probability of the Hypothesis** (red text) points to  $P(H)$ .
- Posterior Probability of the Hypothesis given that the Evidence is True** (blue text) points to  $P(H|E)$ .
- Prior Probability that the evidence is True** (green text) points to  $P(E)$ .

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)}$$

### What makes Naive Bayes a “Naive” algorithm?

Naive Bayes classifier assumes that the features we use to predict the target are independent and do not affect each other. While in real-life data, features depend on each other in determining the target, but this is ignored by the Naive Bayes classifier.

Though the independence assumption is never correct in real-world data, but often works well in practice. so that it is called “**Naive**”.

### Math behind Naive Bays Algorithm

Given a features vector  $X=(x_1,x_2,...,x_n)$  and a class variable  $y$ , Bayes Theorem states that:

$$P(y|X) = \frac{P(X|y) * P(y)}{P(X)}$$

We're interested in calculating the posterior probability  $P(y | X)$  from the likelihood  $P(X | y)$  and prior probabilities  $P(y), P(X)$ .

Using the chain rule, the likelihood  $P(X | y)$  can be decomposed as:

$$\begin{aligned} P(X|y) &= P(x_1, x_2, \dots, x_n | y) \\ &= P(x_1 | x_2, \dots, x_n, y) * P(x_2 | x_3, \dots, x_n, y) \dots P(x_n | y) \end{aligned}$$

but because of the Naive's conditional independence assumption, the conditional probabilities are independent of each other.

$$P(X|y) = P(x_1|y) * P(x_2|y) \dots P(x_n|y)$$

Thus, by conditional independence, we have:

$$P(y|X) = \frac{P(x_1|y) * P(x_2|y) \dots P(x_n|y) * P(y)}{P(x_1) * P(x_2) \dots P(x_n)}$$

And as denominator remains constant for all values, the posterior probability can then be:

$$P(y|x_1, x_2, \dots, x_n) \propto P(y) \prod_{i=1}^n P(x_i|y)$$

The Naive Bayes classifier combines this model with a decision rule. One common rule is to pick the hypothesis that's most probable; this is known as the maximum a posteriori or MAP decision rule.

$$y = \operatorname{argmax}_y P(y) \prod_{i=1}^n P(x_i|y)$$

## How Naive Bays really works:

Let's explain it using an example to make things clear:

Assume we have a bunch of emails that we want to classify as *spam* or *not spam*.

Our dataset has **15 Not Spam** emails and **10 Spam** emails. Some analysis had been done, and the frequency of each word had been recorded as shown below:

	Not Spam	Spam
Dear	8	3
Visit	2	6
Invitation	5	2
Link	2	7
Friend	6	1
Hello	5	4
Discount	0	8
Money	1	7
Click	2	9
Dinner	3	0
Total Words	34	47

**Note:** Stop Words like “the”, “a”, “on”, “is”, “all” had been removed as they do not carry important meaning and are usually removed from texts. The same thing applies to numbers and punctuations.

### exploring some probabilities:

- $P(\text{Dear}|\text{Not Spam}) = 8/34$
- $P(\text{Visit}|\text{Not Spam}) = 2/34$
- $P(\text{Dear}|\text{Spam}) = 3/47$
- $P(\text{Visit}|\text{Spam}) = 6/47$

and so on.

now assume we have the message “*Hello friend*” and we want to know whether it is a spam or not.

so, using Bayes' Theorem

$$P(\text{Not Spam}|\text{Hello Friend}) = \frac{P(\text{Hello Friend}|\text{Not Spam}) * P(\text{Not Spam})}{P(\text{Hello Friend})}$$

ignoring the denominator

$$P(\text{Not Spam}|\text{Hello Friend}) = P(\text{Hello Friend}|\text{Not Spam}) * P(\text{Not Spam})$$

But,  $P(\text{Hello friend} | \text{Not Spam}) = 0$ , as this case (Hello friend) doesn't exist in our dataset, i.e. we deal with single words, not the whole sentence, and the same for  $P(\text{Hello friend} | \text{Spam})$  will be zero as well, which in turn will make both probabilities of being a spam and not spam both are zero, which has no meaning!!

But wait!! we said that the Naive Bayes assumes that ***the features we use to predict the target are independent***.

so,

$$P(\text{Hello Friend}|\text{Not Spam}) = P(\text{Hello}|\text{Not Spam}) * P(\text{Friend}|\text{Not Spam})$$

$$P(\text{Not Spam}|\text{Hello Friend}) = P(\text{Hello}|\text{Not Spam}) * P(\text{Friend}|\text{Not Spam}) * P(\text{Not Spam})$$

$$P(\text{Not Spam}|\text{Hello Friend}) = \frac{5}{34} * \frac{6}{34} * \frac{15}{25} = 0.0155$$

now let's calculate the probability of being spam using the same procedure:

$$P(\text{Spam}|\text{Hello Friend}) = \frac{4}{47} * \frac{1}{47} * \frac{10}{25} = 0.00072$$

so, the message "Hello friend" is not a spam.

**now, let's try another example:**

assume the message "dear visit dinner money money money". It's obvious that it's a spam, but let's see what Naive Bayes will say.

$$P(\text{Not Spam}|\text{dear visit dinner money money money}) \\ = P(\text{dear visit dinner money money money}|\text{Not Spam}) * P(\text{Not Spam})$$

$$P(\text{dear visit dinner money money money}|\text{Not Spam}) = \frac{8}{34} * \frac{2}{34} * \frac{3}{34} * \left(\frac{1}{34}\right)^3 \\ = 3.107 * 10^{-8}$$

$$P(\text{Not Spam}|\text{dear visit dinner money money money}) = 3.107 * 10^{-8} * \frac{15}{25} \\ = 1.864 * 10^{-8}$$

$$P(\text{Spam}|\text{dear visit dinner money money money}) \\ = P(\text{dear visit dinner money money money}|\text{Spam}) * P(\text{Spam})$$

$$P(\text{dear visit dinner money money money}|\text{Spam}) = \frac{3}{47} * \frac{6}{47} * 0 * \left(\frac{7}{47}\right)^3 = 0$$

$$P(\text{Spam}|\text{dear visit dinner money money money}) = 0 * \frac{10}{25} = 0$$

oops!! Naive Bays says that this message is **not a spam**!!!

This happened because the word “*dinner*” does not appear in the *spam* dataset, so that  $P(\text{dinner} | \text{spam}) = 0$ , hence all other probabilities will have no effect.

This is called the **Zero-Frequency Problem**.

and to solve that we can use **Laplace smoothing**.

**Laplace Smoothing** is a technique for smoothing categorical data. A small-sample correction, or pseudo-count, will be incorporated in every probability estimate. Hence, no probability will be zero. this is a way of regularizing Naive Bayes.

Given an observation  $x = (x_1, \dots, x_d)$  from a multinomial distribution with  $N$  trials and parameter vector  $\theta = (\theta_1, \dots, \theta_d)$ , a “smoothed” version of the data gives the estimator:

$$\hat{\theta} = \frac{x_i + \alpha}{N + \alpha d} \quad (i = 1, \dots, d)$$

where the pseudo-count  $\alpha > 0$  is the smoothing parameter ( $\alpha = 0$  corresponds to no smoothing).

### Additive Smoothing

Back to our problem, we are going to pick  $\alpha = 1$ , and ‘d’ is the number of the unique words in the dataset which is 10 in our case.

$$\hat{\theta} = \frac{x_i + \alpha}{N + \alpha d} \quad \alpha = 1, \text{ and } d = 10$$

$$\begin{aligned} P(\text{dear visit dinner money money money} | \text{Not Spam}) &= \frac{8+1}{34+10} * \frac{2+1}{34+10} * \frac{3+1}{34+10} * \left( \frac{1+1}{34+10} \right)^3 \\ &= 1.19 * 10^{-7} \end{aligned}$$

$$\begin{aligned} P(\text{Not Spam} | \text{dear visit dinner money money money}) &= 1.19 * 10^{-7} * \frac{15}{25} \\ &= 7.144 * 10^{-8} = 0.7144 * 10^{-7} \end{aligned}$$

$$\begin{aligned} P(\text{dear visit dinner money money money} | \text{Spam}) &= \frac{3+1}{47+10} * \frac{6+1}{47+10} * \frac{0+1}{47+10} * \left( \frac{7+1}{47+10} \right)^3 \\ &= 0.000000418 = 4.18 * 10^{-7} \end{aligned}$$

$$\begin{aligned} P(\text{Spam} | \text{dear visit dinner money money money}) &= 4.18 * 10^{-7} * \frac{10}{25} \\ &= 1.672 * 10^{-7} \end{aligned}$$

Now it’s correctly classified the message as spam.

## Types of Naive Bayes Classifiers

- **Multinomial:** Feature vectors represent the frequencies with which certain events have been generated by a multinomial distribution. For example, the count how often each word occurs in the document. This is the event model typically used for document classification.
- **Bernoulli:** Like the multinomial model, this model is popular for document classification tasks, where binary term occurrence (i.e. a word occurs in a document or not) features are used rather than term frequencies(i.e. frequency of a word in the document).
- **Gaussian:** It is used in classification and it assumes that features follow a normal distribution.

## Pros and Cons for Naive Bayes

### Pros:

- Requires a small amount of training data. So the training takes less time.
- Handles continuous and discrete data, and it is not sensitive to irrelevant features.
- Very simple, fast, and easy to implement.
- Can be used for both binary and multi-class classification problems.
- Highly scalable as it scales linearly with the number of predictor features and data points.
- When the Naive Bayes conditional independence assumption holds true, it will converge quicker than discriminative models like logistic regression.

### Cons:

- The assumption of independent predictors/features. Naive Bayes implicitly assumes that all the attributes are mutually independent which is almost impossible to find in real-world data.
- If a categorical variable has a value that appears in the test dataset, and not observed in the training dataset, then the model will assign it a zero probability and will not be able to make a prediction. This is what we called the “**Zero Frequency problem**”, and can be solved using smoothing techniques.

## Applications of Naive Bayes Algorithm

- Real-time Prediction.
- Multi-class Prediction.

- Text classification/ Spam Filtering/ Sentiment Analysis.
- Recommendation Systems.

## **Naive Bayes Classifier using Python**

Let's implement the Naive Bayes algorithm using python. This dataset represents patients who were diagnosed with Kyphosis and had a corrective operation. The first column "Kyphosis" represents whether the surgery was successful and corrective of the curvature.

Dataset features: "Age", "Number", "Start"

Target variable: "Kyphosis"

- Was the condition absent or present after the surgery? Represented by "absent" or "present".  
This is our target.
- Age is represented in months as this data is pediatric data
- Number is the number of vertebrae operated on during the operation.
- Start is the most superior number in the vertebral column that was operated on.

Let's follow the steps below.

1. Load dataset
2. Data Pre-Processing
3. Train a model using Decision Tree



4. Predict test results
5. Calculate test accuracy of the model

## Load dataset

```
// import necessary librariesimport pandas as pd
import numpy as np// load datasetdata =
pd.read_csv('kyphosis.csv')data.head()
```

	Kyphosis	Age	Number	Start
0	absent	71	3	5
1	absent	158	3	14
2	present	128	4	5
3	absent	2	5	1
4	absent	1	4	15

## Data Pre-Processing

```
// assign dependent variables without target variablex =
data.drop('Kyphosis', axis=1)// assign target variabley =
data['Kyphosis']// divide data into training and testing setfrom
sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.3)x_train.shapeOUTPUT: (56, 3)x_test.shapeOUTPUT: (25,
3)
```

## Train a model using Naive Bayes

```
// train model using training setfrom sklearn.naive_bayes import
GaussianNB
model = GaussianNB()
model.fit(x_train, y_train)
```

## Predict test results

```
pred = model.predict(x_test)print(pred)OUTPUT:  
array(['present', 'absent', 'present', 'absent', 'present',  
      'absent',      'absent', 'absent', 'present', 'absent', 'absent',  
      'absent',      'absent', 'absent', 'present', 'absent', 'absent',  
      'present',      'absent', 'absent', 'absent', 'absent', 'absent',  
      'absent',      'absent'], dtype=object)
```

## Calculate test accuracy of the model

```
from sklearn.metrics import accuracy_score  
accuracy_score(y_test, pred)OUTPUT: 0.84// create a confusion  
matrixfrom sklearn.metrics import confusion_matrix  
confusion_matrix(y_test, pred)OUTPUT:array([[18,  3],      [ 1,  
      3]], dtype=int64)
```